EXHIBIT 13

Documentation / HTTP Live Streaming API Changes: None

HTTP Live Streaming

Send audio and video to iOS, tvOS, and macOS devices.

Overview

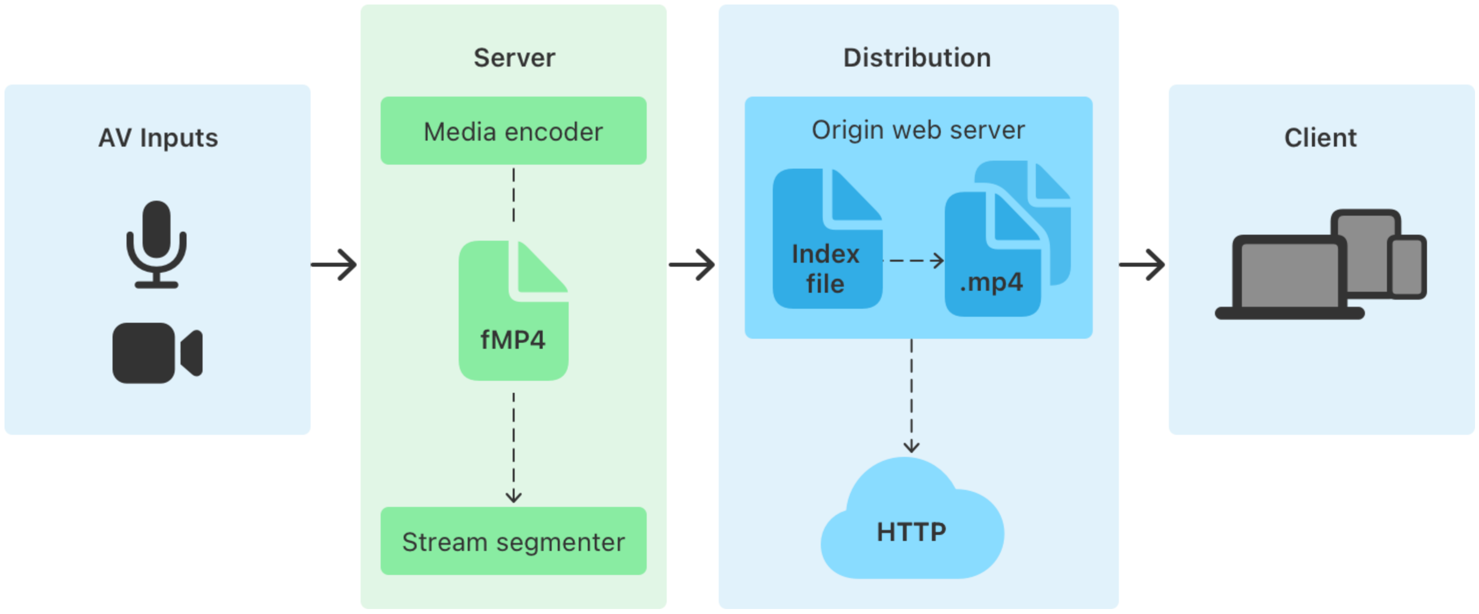
HTTP Live Streaming (HLS) sends audio and video over HTTP from an ordinary web server for playback on iOS-based devices—including iPhone, iPad, iPod touch, and Apple TV—and on desktop computers (macOS). Using the same protocol that powers the web, HLS deploys content using ordinary web servers and content delivery networks. HLS is designed for reliability and dynamically adapts to network conditions by optimizing playback for the available speed of wired and wireless connections.

HLS supports the following:

- Live broadcasts and prerecorded content (video on demand, or VOD)
- Multiple alternate streams at different bit rates
- Intelligent switching of streams in response to network bandwidth changes
- Media encryption and user authentication

Encode and deliver streaming media

The following figure shows the three components of an HTTP Live Stream: the server component, the distribution component, and the client software.



In a typical configuration, a hardware encoder takes audio-video input, encodes it as HEVC video and AC-3 audio, and outputs a fragmented MPEG-4 file or an MPEG-2 transport stream. A software stream segmenter then breaks the stream into a series of short media files, which are placed on a web server. The segmenter also creates and maintains an index file containing a list of the media files. The URL of the index file is published on the web server. Client software reads the index, then requests the listed media files in order and displays them without any pauses or gaps between segments.

Prepare media with the server component

The server component is responsible for taking input streams of media and encoding them digitally. It encapsulates them in a format suitable for delivery and prepares the encapsulated media for distribution.

For live events, the server requires a media encoder, which can be off-the-shelf hardware, and a way to break the encoded media into segments and save them as files, which can either be software such as the media stream segmenter provided by Apple or part of an integrated third-party solution.

Deliver files with the distribution component

The distribution system is a web server or a web-caching system that delivers the media files and index files to the client over HTTP. No custom server modules are required to deliver the content, and typically very little configuration is needed on the web server. To actually deploy HTTP Live Streaming, you need to create either an HTML page for browsers or a client app to act as a receiver. You also need the use of a web server and a way to encode live streams as fragmented MPEG-4 media files containing HEVC or H.264 video, and AAC or AC-3 audio.

Access media through client software

Client software is responsible for determining the appropriate media to request, downloading those resources, and then reassembling them so that the media can be presented to the user in a continuous stream. For the rules governing the interaction between an HLS player and its server, see HTTP Live Streaming 2nd Edition.

Apple provides several frameworks that support HTTP Live Streaming, including AVKit, AVFoundation, and WebKit. Support has been available since iOS 3.0 and Safari 4.0, so there's no need to develop your own client software.

However, if you do develop your own client software, begin by fetching the index file using a URL that identifies the stream. The index file, in turn, specifies the location of the available media files, decryption keys, and any alternate streams available. For the selected stream, download each available media file in sequence. Each file contains a consecutive segment of the stream. Once it has a sufficient amount of data downloaded, present the reassembled stream to the user.

Important

Your client is responsible for fetching any decryption keys, authenticating or presenting a user interface to allow authentication, and decrypting media files as needed.

Continue this process until your client encounters the `EXT-X-ENDLIST` tag in the index file. If no `EXT-X-ENDLIST` tag is present, the index file is part of an ongoing broadcast. During ongoing broadcasts, load a new version of the index file periodically. Look for new media files and encryption keys in the updated index and add these URLs to the playback queue.

Topics

Essentials

Deploying a Basic HTTP Live Streaming (HLS) Stream

Create a basic webpage to deliver HLS.

Preparing Audio for HTTP Live Streaming

Encode your media properly to ensure synchronized audio and video playback.

Stream creation

Learn to create a stream for ingestion by apps enabled with HTTP Live Streaming. Ensure correct playlist formatting and adherence to guidelines.

Example playlists for HTTP Live Streaming

View and compare playlists for different HLS applications.

About the EXT-X-VERSION tag

Find the protocol version that corresponds with the HLS features your app supports.

Tool usage and validation

Use the provided tools to segment your stream, create multivariant playlists, and verify the output of your own tools.

Using Apple's HTTP Live Streaming (HLS) Tools

Segment your video stream and create media playlists for successful transmission with Apple's provided tools.

Specifications and other documents

HTTP Live Streaming (HLS) authoring specification for Apple devices

Learn the requirements for live and on-demand audio and video content delivery using HLS.

About the Common Media Application Format with HTTP Live Streaming (HLS)

Learn the Common Media Application Format as it applies to HLS.

Enabling Low-Latency HTTP Live Streaming (HLS)

Add Low-Latency HLS to your content streams to maintain scalability.

Links to additional specifications and videos

Review additional specifications and documents.

Videos about HLS

Review informational videos about HTTP Live Streaming.

Providing metadata for xHE-AAC video soundtracks

Ensure volume normalization by including metadata for loudness and dynamic range control.

Adjusting anchor loudness

Adjust anchor loudness when measurements of speech-gated loudness for a full mix may be inaccurate, such as when speech activity is low.

Providing JavaScript Object Notation (JSON) chapters

Prepare JSON chapters for HTTP Live Streaming.

Article

Video on Demand playlist construction

Understand the basic composition for a Video on Demand playlist.

Overview

For Video on Demand (VOD) sessions, media files are available representing the entire duration of the presentation. The index file is static and contains a complete list of URLs to all media files created since the beginning of the presentation. This kind of session allows the client full access to the entire program.

Example

This code is an example of a Video on Demand playlist:

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:4
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
http://example.com/movie1/fileSequenceA.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceB.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceC.ts
#EXTINF:9.0,
http://example.com/movie1/fileSequenceD.ts
#EXT-X-ENDLIST
```

These are the tags used in the Video on Demand playlist example:

EXTM3U: Indicates that the playlist is an extended M3U file. This type of file is distinguished from a basic M3U file by changing the tag on the first line to EXTM3U. All HLS playlists must start with this tag.

EXT-X-PLAYLIST-TYPE: Provides mutability information that applies to the entire playlist file. This tag may contain a value of either EVENT or VOD. If the tag is present and has a value of EVENT, the server must not change or delete any part of the playlist file (although it may append lines to it). If the tag is present and has a value of VOD, the playlist file must not change.

EXT-X-TARGETDURATION: Specifies the maximum media-file duration.

EXT-X-VERSION: Indicates the compatibility version of the playlist file. The playlist media and its server must comply with all provisions of the most recent version of the IETF Internet-Draft of the HTTP Live Streaming specification that defines that protocol version.

EXT-X-MEDIA-SEQUENCE: Indicates the sequence number of the first URL that appears in a playlist file. Each media file URL in a playlist has a unique integer sequence number. The sequence number of a URL is higher by 1 than the sequence number of the URL that preceded it. The media sequence numbers have no relation to the names of the files.

EXTINF: A record marker that describes the media file identified by the URL that follows it. Each media file URL must be preceded by an EXTINF tag. This tag contains a duration attribute that's an integer or floating-point number in decimal positional notation that specifies the duration of the media segment in seconds. This value must be less than or equal to the target duration.

Important

Always use floating-point EXTINF durations (supported in protocol version 3). This allows the client to minimize round-off errors when seeking within the stream.

EXT-X-ENDLIST: Indicates that no more media files will be added to the playlist file.

The VOD playlist example above uses full pathnames for the media file playlist entries. While this is allowed, using relative pathnames is preferable. Relative pathnames are more portable than absolute pathnames and are relative to the URL of the playlist file. Using full pathnames for the individual playlist entries often results in more text than using relative pathnames. Here's the same playlist with relative pathnames:

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
```

```
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:4
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
fileSequenceA.ts
#EXTINF:10.0,
fileSequenceB.ts
#EXTINF:10.0,
fileSequenceC.ts
#EXTINF:9.0,
fileSequenceD.ts
#EXT-X-ENDLIST
```

See Also

Basic playlists

Live Playlist (sliding window) construction

Understand the basic composition for a live session playlist.

Event playlist construction

Learn about the basic composition of an event session playlist.

Creating a Multivariant Playlist

Offer multiple playlist files to provide different encodings of the same content.

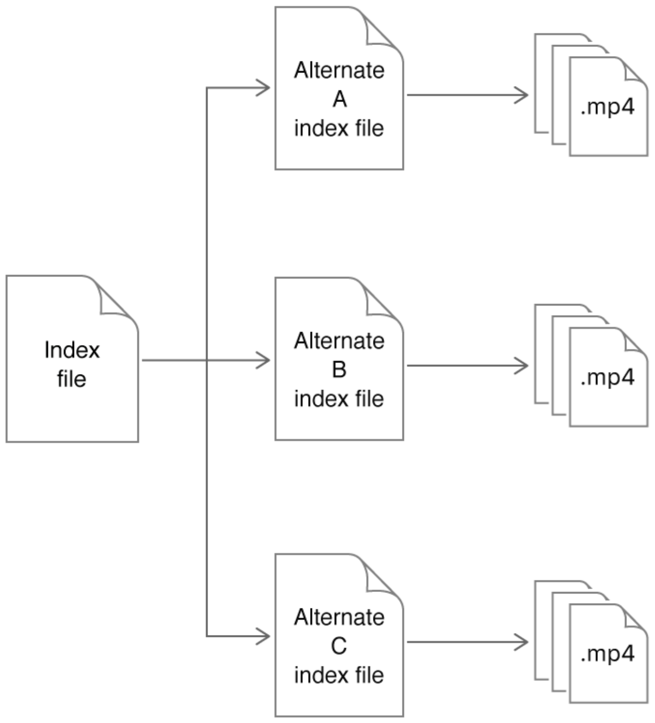
Article

Creating a Multivariant Playlist

Offer multiple playlist files to provide different encodings of the same content.

Overview

The Multivariant Playlist describes all of the available variants for your content. Each variant is a version of the stream at a particular bit rate and is contained in a separate playlist. The client switches to the most appropriate variant based on the measured network bit rate. The client's player is tuned to minimize stalling of playback, to give the user the best possible streaming experience.



A Multivariant Playlist isn't re-read. Once the client has read the playlist, it assumes the set of variations isn't changing. The stream ends as soon as the client sees the `EXT-X-ENDLIST` tag on one of the individual variant playlists.

Define variants

The following example shows a Multivariant Playlist that defines five different variants.

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/low/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=240000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/lo_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/hi_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/high/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=64000,CODECS="mp4a.40.5"
http://example.com/audio/index.m3u8
```

The tags used in the playlist example are:

`EXTM3U`

Indicates that the playlist is an extended M3U file. This type of file is distinguished from a basic M3U file by changing the tag on the first line to `EXTM3U`. All HLS playlists must start with this tag.

`EXT-X-STREAM-INF`

Indicates that the next URL in the playlist file identifies another playlist file.

The `EXT-X-STREAM-INF` tag has the following parameters:

`AVERAGE-BANDWIDTH`

(Optional, but recommended) An integer that represents the average bit rate for the variant stream.

`BANDWIDTH`

(Required) An integer that is the upper bound of the overall bit rate for each media file, in bits per second. The upper bound value is calculated to include any container overhead that appears or will appear in the playlist.

`FRAME-RATE`

(Optional, but recommended) A floating-point value that describes the maximum frame rate in a variant stream.

`HDCP-LEVEL`

(Optional) Indicates the type of encryption used. Valid values are `TYPE-0` and `NONE`. Use `TYPE-0` if the stream may not play unless the output is protected by HDCP.

`RESOLUTION`

(Optional, but recommended) The optional display size, in pixels, at which to display the video in the playlist. This parameter should be included for any stream that includes video.

`VIDEO-RANGE`

(Required depending on encoding) A string with valid values of SDR or PQ. If transfer characteristic codes 1, 16, or 18 aren't specified, then this parameter must be omitted.

`CODECS`

(Optional, but recommended) A quoted string containing a comma-separated list of formats, where each format specifies a media sample type that's present in a media segment in the playlist file. Valid format identifiers are those in the ISO file format name space defined by RFC 6381.

Note

While the `CODECS` parameter is optional, every `EXT-X-STREAM-INF` tag should include the attribute. This attribute provides a complete list of codecs that are necessary to decode a particular stream. It allows the client to distinguish between variants that are audio only and those that have both audio and video. The client then makes use of this information to provide a better user experience when switching streams.

See Also

Basic playlists

Video on Demand playlist construction

Understand the basic composition for a Video on Demand playlist.

Live Playlist (sliding window) construction

Understand the basic composition for a live session playlist.

Event playlist construction

Learn about the basic composition of an event session playlist.